Analysis of a Web User Interface for Mathematics: Experiences with Integral Queries for TILU (Table of Integrals Look Up)

Richard Fateman
Timothy James
Computer Science Division
University of California, Berkeley

Abstract

What can we learn from the range of over 7000 queries made to TILU, a symbolic integration problem-solving server [1] during the course of more than two years? We have saved all queries during this experiment, and based on our analysis, and experimented with improvements in the computer-human interaction components of our web server. We believe our experience will be useful in the design of similar servers that require human input of mathematics.

1 Introduction

TILU[1] [2] serviced 7363 queries from the date it started running through January 26, 1998[2] 6995 of these queries were successful, meaning TILU gave back an answer as opposed to stalling without a response. Not all these answers were useful, but then, not all the questions made sense.

The base program has undergone some change during this period, and there was never a controlled audience: it was made available over the internet immediately. We have received complaints that this was not a carefully controlled experiment, and that is certainly true. This paper nevertheless reports *experiences* with this setup. The program and the web page interface has evolved subject to three pressures:

- The need to handle bad input: Numerous ill-formed inputs changed (mostly lowered) our expectations of what users could be expected to enter into a web form. We found certain categories of gross input format errors could be warded off simply by *eliminating options*. As a simple example, we initially asked the user to type in the variable of integration. We found instances in which the user apparently did not understand this instruction and typed the integrand, or deleted the default entry leaving it blank. Now we simply tell the user we will integrate with respect to x, thereby removing a whole class of errors.

- The need to make the web page easier to use: We looked at our web pages through various browsers set at various resolutions and font sizes, Moving critical parts of the page (buttons) upward within the "first page" became an important priority. Other "web design" issues included making the instructions more prominent.

- The need to make the design and implementation more robust: The underlying program was written as an experiment. Putting it into production uncovered a variety of typographical errors, blunders, and loose ends. Even after making the program relatively secure against unsyntactic input, it took more effort to make the program relatively robust against web communication problems. For errors related to browser connections, we now record such problems but return the program to a standard state. Previously we had established the practice of accumulating suspended lightweight processes for debugging. We would run out of the processes since we began to leave the system running unattended for weeks at a time.

2 Who has been using TILU?

The data that we analyzed consisted of those queries that had domain names associated with them. Many of the queries came from sites that did not send their hostname, but instead sent their IP address, which we do not log. This makes it difficult to know where they originated, although it remains possible for us to analyze these entries at a later time.

There were 3500 successful queries that had domain names associated with them. Of these 3500, 349 were from the Berkeley.EDU domain. Many of these were associated with our own tests and have been removed from the analysis. This brings the number of queries to analyze down to 3151. Of these 3151 queries, we selectively analyzed in more detail, 1015.

Our most popular non-Berkeley sites were as follows:

We partitioned the queries into those from domains with fewer than 5 queries total. and those from domains with at least 5 queries total.

We believed that domains with fewer than 5 queries would be more prone to grammatically incorrect requests, and that sites somehow "learned" the syntax from experience. Our speculation was correct, but experience was only a modest predictor of success.

Under these two groups, we analyzed most single domains that had fewer than 5 requests. For those domains that had at least 5 requests, we selected a few to analyze in detail.

[1] `http://http.cs.berkeley.edu/~fateman/htest.html`

[2] Although TILU's usual host machine has not always been up, the traffic has increased so that it has fielded an average of 291 queries per month between April 1998 and May 1999.

Connects	Location
1008	Domain .edu (excluding Berkeley)
62	MIT
61	Stanford
53	Purdue
43	UCSB
653	Domain .com
49	Compuserve
260	Domain .net
53	uu.net
32	att.net
81	.gov
	Domain: non USA
202	France
189	Germany
118	Sweden
77	Finland
71	Canada
60	Great Britain
53	Switzerland
51	The Netherlands
43	Australia

Table 1: Major sources of queries

The results are sorted by "frequent" and "rare" The "frequent" results come from domains with at least five requests. The "rare" results are those that come from domains that had fewer than five requests. Since our characterizations are done "by humans" we may have misunderstood the nature of the users' misunderstandings, but our results are as follows.

Some explanations are needed: GIGO stands for "Garbage In, Garbage Out".

"*" means that it was a correctly formed request, but TILU did not find the answer. "good" means we solved the problem that was posed: correct input and correct output.

Our web site gave users the choice of using Lisp syntax or Mathematica (MMA) syntax.[3]

[3]We liberalized the "Mathematica" language category in various ways, for example, to permit `sin[x]` even though `Sin[x]` is required by that system. Although we did allow `sin(x)`, the Mathematica interpretation as the product of a constant named `sin` and the variable `x` was undoubtedly wrong to most, if not all, users.

	freq	rare
good (*)	240	109
GIGO - bizarre input	106	45
GIGO - used the wrong language (LISP instead of MMA or vice versa)	78	28
GIGO - used "mathese"	61	52
TILU error - failure of (*)	92	31
TILU couldn't find integral (*)	54	18
user attempted def. int. (*)	16	10
"integral not found" (*)	16	7
User put request in variable of integration box	9	6
User left integrand blank	7	14
Miscellaneous (*)	10	5
Totals	689	326

Table 2: Results

Many people ignored our pleas to use some well-defined syntax: they appeared immune to any advice. Our conclusion is that one should not expect web users to read the online equivalent of even the briefest syntax manual. Of course anyone using such a system must provide mathematical input in some form and thus, one might reasonably expect that each user had to come to grips with the problem. We did not realize the extent to which humans assumed that there was a human-like intelligence on the other end to figure out any old typed material. We call the language used by people who wished us to figure out their made-up syntax "mathese". One web visitor even sent us, via the email comments route, a scanned image (in JPEG form) of a scrap of paper. He asked us for help with his extra-credit calculus "project."

Some people insisted on typing "integrate(...)" instead of just the integrand.

Some users attempted *definite* integrals of simple problems, such as $\sin(x)$ between 0 and 1. We deliberately didn't handle these: we have a short table of "difficult" definite integrals, but we don't use the fundamental theorem of calculus to substitute values into indefinite integrals. If we intended to be more useful, we would obviously need to do this too. Although *we* did not intend TILU to be a substitute computer algebra system, the users had no such prejudice!

When the integration problem was successfully posed and yet TILU could not find the requested integral, there were two common explanations.

- Some failures were caused when a relatively simple problem was submitted that could easily be done by table lookup or (more likely) by a simple algorithmic transformation followed by table lookup, *but not directly in the form given*. These were typically problems that could be done by substitution or the simple "derivative divides" method that can be implemented in a few pages of code, given appropriate "symbolic mathematics" support (It requires that one be able to compute derivatives and divide polynomials. As it happens, in the on-line TILU we have removed these facilities!)
- The second type of failure at this stage were typically caused by requests to integrate expressions that were overly complicated and would likely *not* be done by any computer system: they simply fail to have a closed form symbolic solution. While some may involve functions unknown to TILU, or symbolic constants, others are arrangements of functions that are suitable for numerical quadrature.

Some observations from these tables: For most domains, about 34% of the problems are solved; If there are few interactions from a domain there is nearly a 1 in 2 chance of the problem being erroneously formed. For the frequent users, over 62% are well formed. Subsequent to our data gathering, we encountered some determined users who learned the syntax and checked their calculus homework: their success rates were quite high but atypical.

Some anecdotal observations:

Probably the most common question was a request to integrate `(sin x)`. Some people visiting saw this work and then never came back!

People often attempt to integrate the same problem over and over, sometimes with minor variations in the input, but sometimes without any change at all. Sometimes we found the same *successful* problem posed repeatedly, too. It could be that users are uncomfortable with their browser software and end up re-submitting the same problem by accident, but this seems unlikely to explain all the input.

3 Improving the user interface

We remain somewhat cautious in our analysis, since our experimental data is not all from the same experiment! The early users had our early interface! We think the changes listed below have been helpful, but have no controlled experiment to quantify this. It would be possible, by viewing the progression of errors types subsequent to these changes, to try to judge their effects, but we would prefer to believe that we now have a qualitatively better base for design testing in the future, and that our time would be better spent field testing the effectiveness of design changes around this base.

We have already mentioned

- **Forced Choices.** We removed the option that allowed the user to state the variable of integration. We forced the options so that we will integrate with respect to x. This seems to have been a success.
- **The Wakeup Call.** We initially set a default language choice for the user, to save time. By setting a default, visitors were encouraged to ignore the issue of communicating math in an agreed-upon syntax. They made up some "mathese" language perhaps assuming that some clever person would be able to figure it all out. After thinking about this, and finding that more insistent messages pleading with users to "learn the syntax" were being ignored, we removed the default and *made it imperative that the user choose a language*, at least on the initial visit to the TILU web site. The intent was to make a wake-up call to the user: inform the user of the requirement that *some well-defined agreed-upon language must be used* for math. This is a more subtle requirement than might seem at first. (Note that to us, the mathese notation "`sin x`" or "`Sin x`" is simply `sin*x` whose integral is $\sin * x^2/2$). This is not officially an error, though the user may be puzzled by the answer.
- **The Customer Defines the Sale.** We can figure out what the visitor *expects* TILU to do, and respond correspondingly.

4 Great Expectations

In the previous sections we suggested that the expectations of the visitors should be used as a cue to altering our design. Here we reiterate some of the experiences and expand on a selected few.

One way to solve the problem of people attempting "trivial" *definite integrals* between not in our table (or anyone else's!) is by addressing this issue specifically. There are several approaches. One is to detect these and separate them into

1. Numerical integrals: either complain about this or, if computing time is of no import, run a quadrature program. We hesitate in TILU to embark on substantial calculation or end up trying to explain a result that might be low in accuracy. An appropriately phrased response might get us off the hook regarding accuracy, but our policy [2] of returning in 10 milliseconds would clearly have to be abandoned if we included a good numerical quadrature program in TILU. Reliable adaptive programs for numerically integrating any expression that can be typed in, cannot be guaranteed to finish in such a brief time. If we were to do this, we might as well allow arbitrary integral endpoints, rather than the "choose one from a menu" approach which we have used so far.
2. Use the fundamental theorem of calculus. (plug in the limits and subtract) Quadrature is not possible for an integrand with symbolic parameters or symbolic endpoints. If the solution is not stored as a definite integral, but is stored as an indefinite integral, assume the result is free of singularities: evaluate the indefinite integral at upper/lower limits and return the answer, probably with a warning. At the moment, TILU has a rudimentary evaluation program (really a simplifier), but actually does not at the moment know about floating-point evaluation. The underlying Lisp system library supports IEEE floating-point and elementary functions, and so could be fairly easily exploited if necessary; we also have an arbitrary precision float package as well as interval arithmetic support. Since some people seem to think $\sqrt{x}$ can be written as `x^0.5` we have mapped this form to be equivalent to that produced by `Sqrt[x]` or `x^(1/2)`.

Another kind of expectation is that we will be able to parse mathese. That is, we should read `sinx` as $\sin(x)$ and `sin x exp x` as $\sin(x)e^x$ and `exp xx` as e^{x^2}. All these examples are taken from our logs.

We have also encountered the "floating-point fraction" problem in which `x^0.5` is used to denote $\sqrt{x}$. In response we have mapped this form to be equivalent to that produced by `Sqrt[x]` or `x^(1/2)`.

Can we do better? How far can we go in "reading minds"? It is plausible to restrict the use of lexical tokens such as `sin` so that they are always operators: a successful parse that results in an interpretation of `sin` as a constant is worth questioning for several reasons. Variables in conventional mathematics are single characters. Therefore `sin` could even be interpreted as a product of `s*i*n`. Implausible, but we have seen `xx` presumably meaning x^2. If an expression contains additional variables that have an embedded `x`, then there is also a suspicion of error.

Even though we may frown on the use of different brackets such as {} and [] in out expressions, can we adapt to this?

Although we have not hooked this up to TILU, we in fact have a demonstration program that parses an informally specified math syntax (actually TEX) into a formal system. Interestingly, one of the toughest problems is to disambiguate spaces.

It is clear that forcing the user to read some document explaining how he was responsible for bad input is not the road to popularity or usefulness: Defensive programming is not only an issue of defending yourself against invasion or

against misuse, but also an issue of providing a result that makes sense even to someone with incorrect input.

Since one cannot rely on users to read any manual whatsoever, further development of this or any web interface must seriously consider adding to the capabilities of the program those features that a user might conceivably expect to see in the (unread) manual! After all, if you call a program "integrate" is it a bug if it does not integrate some expression? On this basis one could easily say that it is necessary to modify TILU so that it computes numerical quadratures. Granted this could be viewed as orthogonal to the stated objective of writing a program for "Symbolic integration by table lookup."

5 What if we really don't know the answer?

Sometimes we simply don't know the answer, and TILU responds with "failure." This can be ameliorated in at least three ways.

- The first is by adding additional integrals to our table, either from additional published sources or by automatically generating answers by computation and storing them. We are pursuing incorporation of the full complement of integration formulas from a major table of integrals (Gradshteyn and Rhyzik). This work, some of which can be done by "parsing TEX" from the Academic Press CD-ROM has been done with graduate student Eylon Caspi [3]. Methods of generating interesting integral formulas are possible as well [1].

- The second method, which was added in October, 1998, was to take advantage of other "agents" on the world-wide web: We ship unsolved integration problems to `www.integrals.com`, run by Wolfram Research Inc. This method required two new capabilities, neither of which presented much of a technical challenge: converting our internal Lisp form to Mathematica's string notation, and the second was modeling a net interface on the query/response model a browser would use to access WRI's web site.

- A third method would be to include more algorithmic smarts locally: taking the integration programs easily re-usable from other Lisp programs including our own mock-MMA, or the more elaborate system in Macsyma. We have not done this because our initial goal of keeping the interaction cost to the host quite small (10 milliseconds) would be violated.

6 Future Directions

A simple alternative we have considered is attempting to parse mathese more effectively. Clearly *the Mathematica or Common Lisp choice of syntax is a barrier to our customers.* Perhaps Macsyma or Maple syntax would be better: they allow `sin(x)` rather than `Sin[x]`. Yet they require `x*y` for multiplication, rather than `x y` in Mathematica. Defining our own syntax which would parse `sin(x+y)` as a function application but `a(x+y)` as the product `a*(x+y)` is possible, especially since we have faced that prospect in reading integral tables (to pump up Tilu with more data).

A more drastic alternative we are currently pursuing is to force syntactic correctness by providing a web Java page with a template-based input system. The design for this restricted-input system can be similar to that used in Milo or Theorist [4], or the student-oriented front ends that have been constructed for Maple. Of course for this to work fully in our environment the parser must either be in a Web plugin or downloaded as a Java applet.

One prototype of this is being constructed by students Eric Heien and Gifford Cheung. This began in the context of Heien's Java-based graphing calculator applet (similar to, but not nearly as sophicated as, the Macintosh graphing calculator) with the simplest possible input from a template. We are revising this program, currently accessible as `www.torte.berkeley.edu/~eheien` to provide varieties of input comparable to palette/template inputs in computer algebra systems, as well as rapid feedback on incorrect or correct forms as they are being constructed. This interface might be used to generate not only our Lisp-based syntax, but some MathML or XML style result, should that be of interest.

The initial template is a calculator keyboard with the usual keys for arithmetic, trig functions, "inverse" and "hyperbolic". In addition to graphing, one can select functional symbolic outputs such as derivative or (numerical) integral, and "convert to lisp expression". This is far less sophisticated than non-applet alternatives, in particular the work on interactive context-bound templates and notations in Mathematica described by Soiffer [4].

We are also pursuing another interface design, using 2-D handwritten input.

From a historical perspective, there have been numerous "proof of concept" 2-D handwritten mathematics parsers ranging back to 1968, yet they seem to fail to pass some barrier required for more sophisticated input. This is not to deny that free-form input, say by allowing questions to be written on scraps of ordinary paper and submitting them to a computer would be impressive, just that none have yet survived to production status.

We believe that parsing handwritten mathematics incrementally and online is possible, especially if we compel the user to write in certain freshly displayed empty boxes whose positions and initial sizes depend on what has been written so far. (It is also possible to edit the contents of boxes that have already been filled, and it is also possible to use conventional keyboard input to insert characters into boxes, say tabbing from one box to another.).

Our speculation is that a useful system could involve some modest aids to better recognition accuracy: restricted character set for example. The experiences of ordinary handwriting recognition systems (vastly assisted by the Graffiti system, a text input scheme available on Palm Pilot computers) should be relevant.

Clearly a static grid is inadequate if the characters are of varying size and the positions are not grid-aligned. The grid must vary with the context. For example, writing a $\sum$ would create boxes in directions North, South and East for additional handwritten input. By contrast, a simple name may be super/subscripted or merely horizontally concatenated to the East. In the absence of a good stylus input, cycling among the possible boxes could be done by tabbing or mouse pointing, and in our prototype we provide stubs for the handwriting component by allowing ordinary typing into boxes.

This design is still quite preliminary, and given the limited handwriting recognition tools available as Java applets, as well as the difficulty of writing with a mouse, may not be ready for web deployment in the near future.

7 Conclusions

A huge percentage of the people using the World Wide Web are, of course, quite unconcerned with symbolic integration. One might expect that those who are on the prowl for solutions to calculus problems would constitute a select few, and that these elite would be able to successfully type in linearized mathematics notation. Not necessarily so.

Our evidence of the difficulty reinforces the anecdotal evidence that has plagued many projects using computers in mathematical courses, either in labs, self-paced instruction, or as tools for projects or homework.

- The computer algebra system (CAS) presents a barrier to students who must learn syntax for the computer that is different from the already obscure notation used on paper (what is "dx" after all?).
- The CAS, at least for the average non-science student who just wants to get through calculus and will never encourage a cosine again, represents another thing to learn, and we should not assume that it will be perceived as an aid to learning basics.
- We fear that the CAS will get incorrect or in some way unexplainable answers, which the user will blindly accept; if it gets correct answers, even those may require ever more attention to the computer mechanism rather than the mathematics.

Regarding further developments, we may remind the reader that our original intent in building TILU was to provide a math "Internet agent" to be used by traditional, desktop computer-resident algebra systems. TILU was to be fed problems from such systems, over the Internet. With the exception of a one-time only experiment, no one outside our own group took us up on this, however. The practicality of this has been demonstrated by our own use of the Wolfram integrator as an Internet agent, and so perhaps there is hope if others in the computer algebra community can similarly join in.

The point of this paper is a diversion to study the problem of mathematical input on the world-wide web. Perhaps we will see a re-usable and downloadable mathematical input module, either Java-based like Biscotti, or via some well-distributed plugin. Or perhaps we will see a more sophisticated approach using an internet agents that can read and interpret handwritten math.

8 Acknowledgments

Some of this work was supported indirectly by NSF infrastructure grants to the Computer Science Division, EECS, University of California, Berkeley. Timothy James was supported in part by an undergraduate research grant from the College of Engineering.

References

[1] T. H. Einwohner and R. Fateman, "Searching Techniques for Integral Tables," *Proc. of Int'l Symp. on Symbolic and Algebraic Computation*, ISSAC-95 (ACM Press), Montreal, CA, July 1995, 133—139.

[2] Richard J. Fateman, "Network Servers for Symbolic Mathematics," *Proc. of Int'l Symp. on Symbolic and Algebraic Computation*, ISSAC-97 (ACM Press) Maui, Hawaii, July, 1997, 249—256.

[3] Richard J. Fateman and Eylon Caspi, "Parsing TEX into Mathematics" unpublished, `www.cs.berkeley.~fateman/papers/parsing_tex.ps`

[4] Neil M. Soiffer, *The Design of a User Interface for Computer Algebra Systems.* Ph.D thesis, EECS Dept., Univ. of Calif, Berkeley, April, 1991. See also Soiffer's "Mathematical Typesetting in Mathematica," *Proc. of Int'l Symp. on Symbolic and Algebraic Computation*, ISSAC-95 (ACM Press), Montreal, CA, July 1995, 140–149.

9 Appendix

How hard is it to run a socket from Lisp, reading characters and writing them?

Here is the Lisp command that must be run in Allegro Common Lisp to allow connection on port 9015 of the host machine to the biscotti java program, which must by java security conventions, be downloaded from the same machine (currently `torte.cs.berkeley.edu`): `(mp:process-run-function "server" #'biscotti-server 9015)`

The program itself consists of two parts, the server attached to the port, and the connection program. We have omitted the details of the work that is done by COMPUTE-ANSWER which includes parsing the input line (presumably a character string correctly parenthesized for lisp), and producing the answer ready to be printed out to the connection.

```
(defun biscotti-server (port)
  (let ((socket (socket:make-socket
 :connect :passive :local-port port)))
    (unwind-protect
(loop
  (let ((*con* (socket:accept-connection socket)))
    (mp::process-run-function
     "biscotti-process"
     #'biscotti-connection *con*)))
      (close socket))))

(defun biscotti-connection (*con*)
  (ignore-errors
   (format *con*  "~s~%"
   (COMPUTE-ANSWER (read-line *con* nil nil) )
   (close *con*))))
```